

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  /*
6   Problem # 2 Using LU Decomposition-Crout's Method
7  */
8  double dblAbs(double num){
9      num = sqrt(num*num);
10     return num;
11 }
12
13 int main()
14 {
15     double U[13][13], L[13][13];
16     double b[13] = {2, 0, 0, 0, 4, 0, 0, 0, 2, 0, 0, 0, 0};
17     double x[13], y[13], bsol[13];
18     double a0[13][13] = {
19         { 1.0250, -0.4000, 0.0000, 0.0000, -0.6250, 0.0000, 0.0000, 0.0000, 0.0000,
20         0.0000, 0.0000, 0.0000, 0.0000},
21         {-0.4000, 1.9135, -0.2998, 0.0000, 0.0000, -1.0338, -0.1799, 0.0000, 0.0000,
22         0.0000, 0.0000, 0.0000, 0.0000},
23         { 0.0000, -0.2998, 2.1415, -0.4417, 0.0000, 0.0000, -1.4000, 0.0000, 0.0000,
24         0.0000, 0.0000, 0.0000, 0.0000},
25         { 0.0000, 0.0000, -0.4417, 1.0077, 0.0000, 0.0000, 0.0000, -0.5660, 0.0000,
26         0.0000, 0.0000, 0.0000, 0.0000},
27         {-0.6250, 0.0000, 0.0000, 0.0000, 2.0500, -0.8000, 0.0000, 0.0000, 0.0000,
28         -0.6250, 0.0000, 0.0000, 0.0000},
29         { 0.0000, -1.0338, 0.0000, 0.0000, 0.0000, -0.8000, 3.8097, -0.4401, 0.0000,
30         -1.0000, -0.0595, -0.4764, 0.0000},
31         { 0.0000, -0.1799, -1.4000, 0.0000, 0.0000, -0.4401, 4.0492, -0.7067, 0.0000,
32         0.0000, 0.0000, -1.1159, -0.2067},
33         { 0.0000, 0.0000, 0.0000, -0.5660, 0.0000, 0.0000, -0.7067, 2.2161, 0.0000,
34         0.0000, 0.0000, 0.0000, -0.9433},
35         { 0.0000, 0.0000, 0.0000, 0.0000, -0.6250, 0.0000, 0.0000, 0.0000, 1.0250,
36         -0.4000, 0.0000, 0.0000, 0.0000},
37         { 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, -1.0000, 0.0000, 0.0000,
38         -0.4000, 2.0667, -0.6667, 0.0000},
39         { 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, -0.0595, 0.0000, 0.0000,
40         -0.6667, 1.3788, -0.6526, 0.0000},
41         { 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, -0.4764, -1.1159, 0.0000,
42         0.0000, -0.6526, 2.5098, -0.2650},
43         { 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, -0.2067, -0.9433,
44         0.0000, 0.0000, -0.2650, 1.4150},
45     };
46     double a[13][13] = {
47         { 1.0250, -0.4000, 0.0000, 0.0000, -0.6250, 0.0000, 0.0000, 0.0000, 0.0000,
48         0.0000, 0.0000, 0.0000, 0.0000},
49         {-0.4000, 1.9135, -0.2998, 0.0000, 0.0000, -1.0338, -0.1799, 0.0000, 0.0000,
50         0.0000, 0.0000, 0.0000, 0.0000},
51         { 0.0000, -0.2998, 2.1415, -0.4417, 0.0000, 0.0000, -1.4000, 0.0000, 0.0000,
52         0.0000, 0.0000, 0.0000, 0.0000},
53         { 0.0000, 0.0000, -0.4417, 1.0077, 0.0000, 0.0000, 0.0000, -0.5660, 0.0000,
54         0.0000, 0.0000, 0.0000, 0.0000},
55         {-0.6250, 0.0000, 0.0000, 0.0000, 2.0500, -0.8000, 0.0000, 0.0000, 0.0000,
56         -0.6250, 0.0000, 0.0000, 0.0000},
57         { 0.0000, -1.0338, 0.0000, 0.0000, 0.0000, -0.8000, 3.8097, -0.4401, 0.0000,
58         -1.0000, -0.0595, -0.4764, 0.0000},
59         { 0.0000, -0.1799, -1.4000, 0.0000, 0.0000, -0.4401, 4.0492, -0.7067, 0.0000,
60         0.0000, 0.0000, -1.1159, -0.2067},
61         { 0.0000, 0.0000, 0.0000, -0.5660, 0.0000, 0.0000, -0.7067, 2.2161, 0.0000,
62         0.0000, 0.0000, 0.0000, -0.9433},
63         { 0.0000, 0.0000, 0.0000, 0.0000, -0.6250, 0.0000, 0.0000, 0.0000, 1.0250,
64         -0.4000, 0.0000, 0.0000, 0.0000},
65         { 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, -1.0000, 0.0000, 0.0000,
66         -0.4000, 2.0667, -0.6667, 0.0000},
67         { 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, -0.0595, 0.0000, 0.0000,
68         -0.6667, 1.3788, -0.6526, 0.0000},
69         { 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, -0.4764, -1.1159, 0.0000,
70         0.0000, -0.6526, 2.5098, -0.2650},
71         { 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, -0.2067, -0.9433,
72         0.0000, 0.0000, -0.2650, 1.4150},
73     };
74
75     double u0 = 5;
76     int total, col, row, k, i, j, p;
77     double dummy, m;
78
79     total = sizeof(a)/sizeof(a[0][0]); //
80     col = sizeof(a[0])/sizeof(a[0][0]); // Determines the number of rows and columns
81     row = total/col; //
82
83     for(i=0; i<row; i++){

```

```

59     b[i] = b[i]*u0;
60 }
61
62 //Initialize the L and U Matrix to zero
63 for (j=0; j<col; j++){
64     for (i=0; i <row; i++){
65         U[i][j] = 0;
66         L[i][j] = 0;
67     }
68 }
69
70 for (j=0; j<col; j++){
71     U[j][j] = 1;
72     for (i=0; i <row; i++){
73         // Compute for elements of L
74         if (i>=j){
75             L[i][j] = a[i][j];
76             for(k = 0; k < j; k++){
77                 L[i][j] = L[i][j] - L[i][k]*U[k][j];
78             }
79         }
80     }
81     for (i=0; i <row; i++){
82         // Compute for elements of U
83         if (i>j) {
84             U[j][i] = a[j][i];
85             for(k = 0; k<j; k++){
86                 U[j][i] = U[j][i] - L[j][k]*U[k][i];
87             }
88             U[j][i] = U[j][i]/L[j][j];
89         }
90     }
91 }
92
93 for(i=0;i<row;i++){
94     //Forward Substitution
95     y[i] = b[i];
96     for(k = 0; k < i; k++) {
97         y[i] = y[i] - L[i][k]*y[k];
98     }
99     y[i] = y[i]/L[i][i];
100 }
101 for(i=(row-1); i >= 0;i--){
102     //Backward Substitution
103     x[i] = y[i];
104     for(k=(row-1); k > i; k--) {
105         x[i] = x[i] - U[i][k]*x[k];
106     }
107 }
108
109 printf("\nPotential at Each node are:\n");
110 for(i=0;i<row;i++){
111     printf("x[%d] = %0.9lf \n",i+1,x[i]);
112 }
113
114 //Checking:
115 printf("\nB's are:\n");
116 for(i=0;i<row;i++){
117     bsol[i] = 0;
118     for(j=0;j<row;j++){
119         bsol[i] = bsol[i] + a0[i][j]*x[j];
120     }
121     printf("b[%d] : %0.9lf \n",i+1,bsol[i]);
122 }
123
124 }
125
126 return 0;
127 }
128
129

```